Calhoun: Institutional Archive of the Naval Postgraduate School

Calhoun: The NPS Institutional Archive

DSpace Repository

Theses and Dissertations 1. Thesis and Dissertation Collection, all items

1985

Printer multiplexing among multiple Z-100 microcomputer

Choi, Kwang Jun; Lee, Ju Kab

http://hdl.handle.net/10945/21499

Copyright is reserved by the copyright owner.

Downloaded from NPS Archive: Calhoun

DUDLEY KNOX LIBRARY

Calhoun is the Naval Postgraduate School's public access digital repository for research materials and institutional publications created by the NPS community. Calhoun is named for Professor of Mathematics Guy K. Calhoun, NPS's first appointed -- and published -- scholarly author.

Dudley Knox Library / Naval Postgraduate School
411 Dyer Road / 1 University Circle
Monterey, California USA 93943

http://www.nps.edu/library

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA 93943-5002

NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

PRINTER MULTIPLEXING AMONG MULTIPLE Z-100 MICROCOMPUTERS

by

Kwang Jun Choi and Ju Kab Lee

December 1985

Thesis Advisor: Uno R. Kodres

Approved for public release; distribution is unlimited

T226056

REPORT DOCUMENTATION PAGE

a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED

1b. RESTRICTIVE MARKINGS:

a. SECURITY CLASSIFICATION AUTHORITY:

b. DECLASSIFICATION / DOWNGRADING SCHEDULE:

3. DISTRIBUTION / AVAILABILITY OF REPORT: Approved for public release; distribution is unlimited

PERFORMING ORGANIZATION REPORT NUMBER(S):

5. MONITORING ORGANIZATION REPORT NUMBER(S):

a. NAME OF PERFORMING ORGANIZATION: Naval Postgraduate School

6b. OFFICE SYMBOL *(If applicable)*: 52

7a. NAME OF MONITORING ORGANIZATION: Naval Postgraduate School

c. ADDRESS *(City, State, and ZIP Code)*: Monterey, CA 93943-5100

7b. ADDRESS *(City, State, and ZIP Code)*: Monterey, CA 93943-5100

a. NAME OF FUNDING / SPONSORING ORGANIZATION:

8b. OFFICE SYMBOL *(If applicable)*:

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER:

c. ADDRESS *(City, State, and ZIP Code)*:

10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO.

1 TITLE *(Include Security Classification)*: PRINTER MULTIPLEXING AMONG MULTIPLE Z-100 MICROCOMPUTERS (UNCLASSIFIED)

2 PERSONAL AUTHOR(S): Choi, Kwang Jun and Lee, Ju Kab

3a TYPE OF REPORT: Master's Thesis

13b. TIME COVERED: FROM ________ TO ________

14. DATE OF REPORT *(Year, Month, Day)*: 1985 December

15. PAGE COUNT: 79

6 SUPPLEMENTARY NOTATION:

7 COSATI CODES

FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*: Concentrator, Single Board Computer, Z-100, CP/M-85, CP/M-86, I/O Expansion Board, BOOT Process, CONTROL Process, SPOOL Process, Download

9 ABSTRACT *(Continue on reverse if necessary and identify by block number)*

This thesis describes the detailed design and implementation of a printer server in the laboratory environment of sharing resources among multiple Zenith Z-100 microcomputers. The Printer Server System is a controller box which consists of a power supply, a single board computer, and the BLC 8538 eight channel I/O expansion boards. Each Z-100 microcomputer is connected to the controller thru the RS-232C port.

The Printer Server System has three software utilities: BOOT, CONTROL and SPOOL. The BOOT process, resident in the controller, downloads the CONTROL file from any one of multiple Z-100's which is turned on. The CONTROL process allows the printer to be used by any one of multiple Z-100's at a time. The SPOOL process sends the data thru the CONTROL process to the printer or saves the data on the (Continue)

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT: ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS

21. ABSTRACT SECURITY CLASSIFICATION: UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL: Prof. Uno R. Kodres

22b. TELEPHONE *(Include Area Code)*: (408) 626-2197

22c. OFFICE SYMBOL: 52Kr

DD FORM 1473, 84 MAR

83 APR edition may be used until exhausted.
All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

19. ABSTRACT (Continued)

disk file.

Approved for public release; distribution is unlimited.

Printer Multiplexing among Multiple Z-100 Microcomputers

by

Choi, Kwang Jun
Captain, Korea Army
B.S., Korea Military Academy, 1978

and

Lee, Ju Kab
Captain, Korea Army
B.S., Korea Military Academy, 1979

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 1985

Thesis
C44 [illegible]
c.1

ABSTRACT

This thesis describes the detailed design and implementation of a printer server in the laboratory environment of sharing resources among multiple Zenith Z-100 microcomputers. The Printer Server System is a controller box which consists of a power supply, a single board computer, and the BLC 8538 eight channel I/O expansion boards. Each Z-100 microcomputer is connected to the controller thru the RS-232C port.

The Printer Server System has three software utilities: BOOT, CONTROL and SPOOL. The BOOT process, resident in the controller, downloads the CONTROL file from any one of multiple Z-100's which is turned on. The CONTROL process allows the printer to be used by any one of multiple Z-100's at a time. The SPOOL process sends the data thru the CONTROL process to the printer or saves the data on the disk file.

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA 93943-5002

DISCLAIMER

Many terms used in this thesis are registered trademarks of commercial products. Rather than attempt to cite each individual occurence of a trademark, all registered trademarks appearing in this thesis will be listed below, following the firm holding the trademark.

Intel Corporation, Santa Clara, California:

INTEL MULTIBUS iSBC

Digital Research, Pacific Grove, California:

CP/M CP/M-85 CP/M-86

National Semiconductor, Santa Clara, California:

BLC 8538

TABLE OF CONTENTS

I. INTRODUCTION 10
  A. GENERAL DISCUSSION 10
  B. FORMAT OF THESIS 11

II. HARDWARE BASIS OF THE SYSTEM 12
  A. ISBC 86/12A SINGLE BOARD COMPUTER 12
    1. General Description 12
    2. Memory 13
  B. INTEL 8086 14
    1. Major Features of the 8086 14
    2. Registers 14
  C. COMMUNICATION EXPANSION BOARD 16
    1. Mode Register 1 17
    2. Mode Register 2 18
    3. Command Register 18
    4. Status Register 19
    5. SYN1/SYN2/DLE Registers 19
    6. Transmit/Receive Data Holding Registers 20

III. SYSTEM SOFTWARE 22
  A. CP/M-85 OPERATING SYSTEM 22
    1. General Discussion 22
    2. Structure 23
  B. MODIFICATION TO CP/M-85 BIOS 26

IV. IMPLEMENTATION OF THE SYSTEM 28
  A. STRUCTURE OF THE CONCENTRATOR 28
  B. BOOT PROCESS 30
  C. CONTROL PROCESS 30
  D. SPOOL PROCESS 36

1. Function 1 37
2. Function 2 38
3. Function 3 40

V. CONCLUSIONS 41

APPENDIX A: USER MANUAL FOR PRINTER SHARING 42
A. SYSTEM CONFIGURATION 42
B. ACTIVATING THE SYSTEM 42
C. USING THE SPOOL PROGRAM 42
1. Interactive Print 43
2. Saving on a Disk File 43
3. Printing Existing Files 43

APPENDIX B: BOOT PROGRAM 44

APPENDIX C: CONTROL PROCESS 50

APPENDIX D: SPOOL PROCESS 57

LIST OF REFERENCES 77

INITIAL DISTRIBUTION LIST 78

LIST OF FIGURES

2.1 System Configuration 13
2.2 Functional Structure of 8086 15
2.3 Mode Register 1 Format 18
2.4 Mode Register 2 Format 19
2.5 Command Register Format 20
2.6 Status Register Format 21
3.1 Memory Organization of CP/M 25
3.2 Entry Point of List Output Function 27
4.1 Structure of the Concentrator 29
4.2 Logical Flow of BOOT 31
4.3 Algorithm of BOOT 32
4.4 Logical Flow of CONTROL 34
4.5 Algorithm of CONTROL 35
4.6 Logical Flow of SPOOL 37
4.7 Algorithm of SPOOL 39

ACKNOWLEDGEMENTS

We thank God for our health and patience. We are very grateful to a number of people for their help and encouragement.

Most of all, we would like to express our sincere appreciation to our thesis advisor, Professor Uno R. Kodres, for his enthusiastic guidance and support. Without his help and ideas about the program, this thesis could not have been completed. We also would like to say thanks to our second reader, Instructor Linda C. Rawlinson, for her helpful comments and her kindness.

We must thank to our wives, Young Mee Choi and Nam Yoen Lee, and our parents for their sincere help and encouragement during our studying period.

I. INTRODUCTION

A. GENERAL DISCUSSION

In a school or many companies, several personal computers are used on the same floor or the same room. Instead of having a printer with each personal computer, it is economical to share a high quality printer among many users. More generally, it is economical to share other high cost resources such as laser printers, Ethernet local area network controllers, high capacity disc units, among the personal computer users. It is the motivation to share expensive resources that has given rise to this thesis.

To design and implement a system which permits the sharing of a printer among twenty-three users, each user making use of a Zenith Z-100 personal computer, the hardware system, known as the Concentrator, consists of a MULTIBUS based backplane with an independent power supply, an INTEL iSBC 86/12A single board computer, three National Semiconductor BLC 8538 serial I/O expansion boards, all enclosed in a cardcage. The Zenith Z-100 personal computers are connected via their teletype ports(RS232 compatible) to the I/O ports of the Concentrator.

The software system consists of the Control Program,which is down-loaded from a Z-100 at power-up of the Concentrator and any Z-100 system, and which thereafter executes in the Concentrator permitting the shared use of the printer on a first-come first-serve basis.

The remaining software is a utility program, called SPOOL, which operates in the CP/M environment on the Z-100 and permits the user to store his interactive printer data on his disc unit, while the printer is busy printing another user's files.

The Control Program in the concentrator is implemented in assembly language. The CP/M operating system in the

Z-100 is modified by modifying the Basic Input/Output System (BIOS) of CP/M-85 for use for communication with the Concentrator.

The SPOOL program, also written in assembly code, allows the users to share the high speed printer in a batch mode, that is, each user prints out the files he desires, while any other user who wishes to make interactive use of the printer at the same time can save the information in his disk file for later batch printing. This permits an economical usage of a high cost resource without the inconvenience of long waits due to interactive use of the printer.

B. FORMAT OF THESIS

Chapter I gives an overview of this thesis. It also provides the general concepts and the construction of our system and explains why this thesis has practical value.

Chapter II explains the hardware basis of the system. Detailed information is given about all major hardware components which are used to construct the printer sharing system which is implemented in this thesis.

Chapter III describes the general structure of the CP/M operating system. The standard CP/M-85 operating system is discussed in moderate detail. Also covered is the modification of the CP/M-85 necessary to carry out the project.

Chapter IV explains the role and the structure of the Concentrator. Three utility programs, BOOT, CONTROL, and SPOOL, are explained in detail.

Chapter V summarizes the advantage of this system and indicates a direction for future research for microcomputer communication and networking using the local area network, Ethernet.

II. HARDWARE BASIS OF THE SYSTEM

As stated in the introduction, the most important hardware components of this system are the iSBC 86/12A single board computer, the I/O expansion board, the multiple Z-100 microcomputers and the printer. Figure 2.1 depicts the interconnection of these components as they exist at this time. In the paragraphs which follow, a description of each component and its role in the overall system, is explained. A detailed description of each component can be found in the cited references.

A. ISBC 86/12A SINGLE BOARD COMPUTER

1. General Description

The iSBC 86/12A single board computer, which is a member of Intel's complete line of 8-bit and 16-bit single board computer products, is a complete computer system on a single printed-circuit assembly. This board includes a 16-bit Central Processing Unit(CPU), up to 32K bytes of Erasable Programmable Read Only Memory(EPROM), 64K bytes of Random Access Memory (RAM), a serial communications interface, three programmable parallel I/O ports, programmable timers, priority interrupt control, MULTIBUS interrupt control logic, and bus expansion drivers for interfaces with other MULTIBUS interface-compatible expansion boards. The systems view, however, is an abstract model, as systems exist only in the mind of the analyst. [Ref. 4: 1.1 - 1.3].

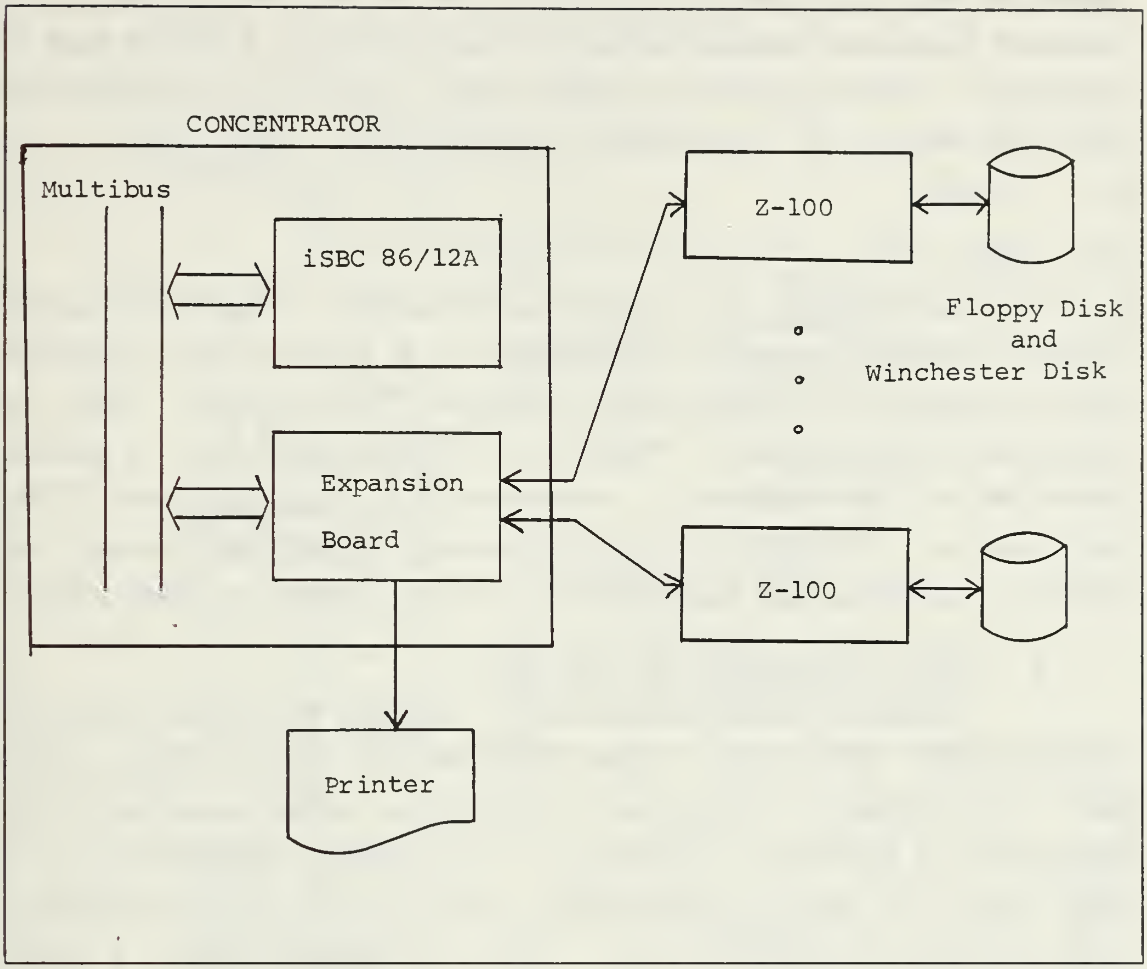


Figure 2.1 System Configuration.

2. Memory

The iSBC 86/12A board includes 64K bytes of read/write memory composed of sixteen 2117 Dynamic RAM chips and an 8202 RAM controller.

Four IC sockets are provided for user installation of EPROM chips, and jumpers are provided to accommodate either 2K, 4K, or 8K chips. The EPROM address space is located at the top of the 1-megabyte memory space because the 8086 CPU branches to FFFF0 after a reset.

This single board computer is used as the controller in the system. The control process of the Concentrator is loaded into the memory of this iSBC 86/12A single board computer at the time of initialization of the system. The control process communicates to the multiple Z-100's and the printer, and receives or sends data from or to the devices. For the detail of the memory organization, see [Ref. 4: pp 4.2 - 4.3].

B. INTEL 8086

The Intel 8086 is a high performance, general purpose 16-bit microprocessor. The 8086 is a full 16-bit processor with respect to both its internal structure, and its external connections. Refer to Figure 2.2 for a general overview of its internal structure and organization. This section is intended to give general knowledge about this device. A detailed description can be found in [Ref. 3].

1. Major Features of the 8086

In this section, we shall discuss the major features of the 8086 CPU chip. The 8086 has a 20-bit-wide address bus, providing it with the capability of addressing a full megabyte of memory. However, the address registers of the 8086 chip are only sixteen bits wide. This is equivalent to only sixty four kilobytes. This processor uses a method called segmentation to allow smooth access to the whole megabyte of address space. The twenty bit physical address is formed, by adding a sixteen bit offset to a twenty bit address whose most significant sixteen bits come from a sixteen bit segment register.

2. Registers

The 8086 contains fourteen 16-bit registers. Some of these belong to the EU(Execution Unit) and others belong to the BIU(Bus Interface Unit). The EU registers tend to be general-purpose registers, whereas the BIU registers tend to be used for addressing. There are eight 16-bit general purpose registers. Four of these are byte or word addressable and are referred to as the 'data group'. The remaining four are only word addressable and are referred to

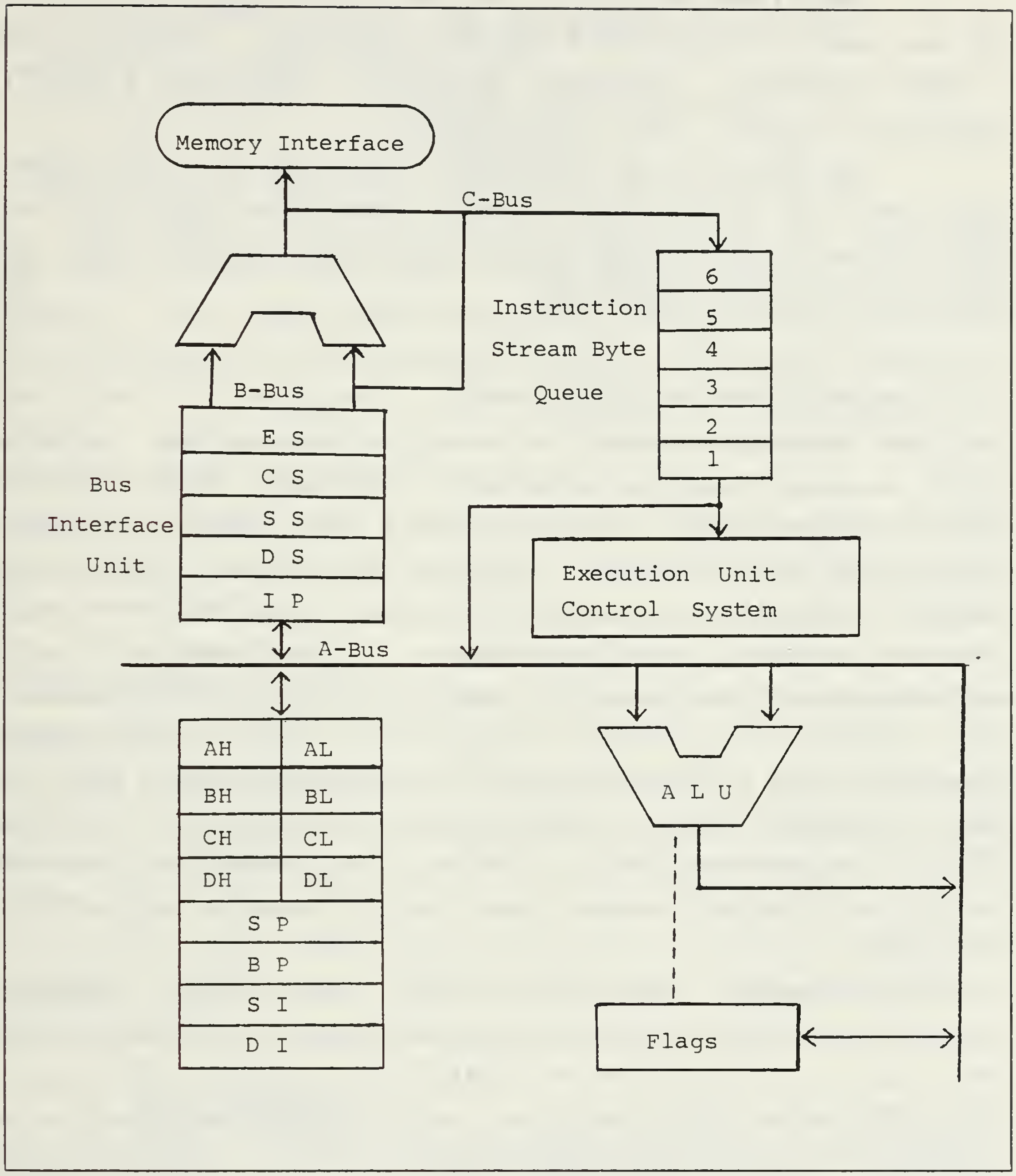


Figure 2.2 Functional Structure of 8086.

as the pointer and index register, SP, BP, SI, and DI, which cannot be further subdivided.

SP is the stack pointer, BP is the base pointer and SI and DI are named the source and destination index register, respectively.

The flags register is sixteen bits wide and consists of nine usable status bits for the processor. These include : zero flag(ZF), direction flag(DF), interrupt flag(IF), overflow flag(OF), and trap flag(TF).

The BIU has the remaining registers : four segment register, CS, DS, SS, and ES. These stand for the code, data, stack, and extra segment register respectively, and one instruction pointer(IP).

C. COMMUNICATION EXPANSION BOARD

This expansion board is used to connect the multiple Z-100 computers and the printer to the single board computer of the Concentrator. We can connect from fifteen to twenty three Z-100 microcomputers with two or three BLC 8538 eight channel communication expansion boards. The BLC 8538 eight channel communication expansion board provides fully programmable synchronous of asynchronous serial communication channels with RS232C interfaces to expand system serial communications capability. Each individual channel provides two interrupt requests with distinct priorities. The BLC 8538 is compatible with all National Semiconductor Serial 80 cardcages, back planes and microcomputer system bus structures.

The universal synchronous and asynchronous receiver/ transmitter(USART) control logic block gates the data on the data bus to the appropriate USART reset pins. Writing to the USART reset logic register resets each USART displaying a logic one on its corresponding data line. Two 74LS08 and gate devices gate the data bus to the appropriate USART reset pins. Data lines zero thru seven(DA0-DA7) are logically ANDed with the reset I/O line. Writing to the reset register causes each set data bit in the USART control logic to reset its corresponding USART.

A USART reset causes the following :

a) Resets status bits to zero,
b) Disables the transmitter and receivers,
c) Clears any interrupt requests,
d) Clears the mode, status and command registers.

The USART's function is to transmit eight (or less) bits of data on one signal wire in a serial fashion to another USART. The other USART assembles the serial bit pattern into an eight bit value transferrable to the data lines of a data bus of a computer. Each USART uses four address locations : one as a data register, one as a control register, one as mode selection register, and one as a status register. The 2651 has nine 8-bit registers that can be written to and/or read from by the system CPU through the data bus D7-D0. The nine 8-bit registers are mode register 1, mode register 2, command register, status register, SYN1/SYN2/DLE registers, and transmit/receive data holding registers.

The 2651 USARTs must be programmed by the system CPU before transmitting or receiving data. Programming the USART consists of four steps. First, the user should reset the USART. Second, the user writes to the Mode Register 1. On the third step, the user writes to the Mode Register 2. Finally, the user writes to the Command Register. The registers are described in detail below.

1. Mode Register 1

This register sets the mode(synchronous or asynchronous), parity use and type, character length, and other parameters. The register format is shown in Figure 2.3. In our program, the programming code will be 01001110B(4EH).

2. Mode Register 2

This register sets the baud rate and the external/internal characteristic of the transmitter and receiver clocks. The register format is shown in Figure 2.4. The programming code will be 00111110B(3EH).

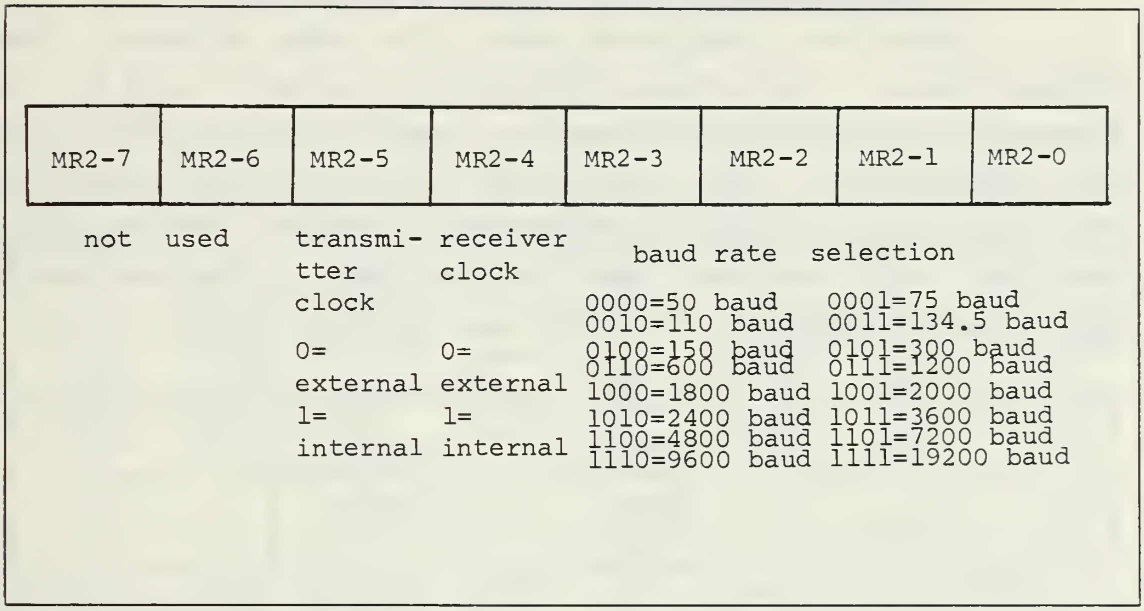
MR1-7	MR1-6	MR1-5	MR1-4	MR1-3	MR1-2	MR1-1	MR1-0
sync: no. of syn chars 0=double syn 1=single syn	sync: transparency control 0=normal 1=transparent	parity type 0=odd 1=even	parity control 0= disabled 1= enabled	character length 00=5 bits 01=6 bits 10=7 bits 11=8 bits		mode and baud rate factor* 00=sync 1x rate 01=async 1x rate 10=async 16x rate 11=async 64x rate	
async:stop bit length 00=invalid 01=1 stop bit 10=1½ stop bits 11=2 stop bits							

*baud rate factor in asynchronous mode applies only if external clock is selected. Factor is 16x if internal clock is selected.

Figure 2.3 Mode Register 1 Format.

3. Command Register

This register controls the DTR (Data Terminal Ready) and RTS (Request to Send Data) outputs, operating mode, transmit control enable and receive control enable. The register format is shown in Figure 2.5. The command register can be written to, but not read from. The programming code will be 00000111B(07H).



MR2-7	MR2-6	MR2-5	MR2-4	MR2-3	MR2-2	MR2-1	MR2-0
not	used	transmitter clock 0= external 1= internal	receiver clock 0= external 1= internal	baud rate selection			

0000=50 baud 0001=75 baud
0010=110 baud 0011=134.5 baud
0100=150 baud 0101=300 baud
0110=600 baud 0111=1200 baud
1000=1800 baud 1001=2000 baud
1010=2400 baud 1011=3600 baud
1100=4800 baud 1101=7200 baud
1110=9600 baud 1111=19200 baud

Figure 2.4 Mode Register 2 Format.

4. Status Register

The status register indicates the state of the TxRDY (Transmitter Ready Interrupt) and RxRDY (Receiver Ready Interrupt) outputs and the DCD (Data Carrier) and DSR (Data Set Ready) inputs. The register format is shown in Figure 2.6. The status register can be read from, but not written to. If the Transmitter is ready, the code will be 00000001B(01H). If the Receiver is ready, the code will be 00000010B(02H).

5. SYN1/SYN2/DLE Registers

If synchronous operation is chosen, these three registers must be loaded to provide characters for synchronization, idle, fill, and data transparency. The use of these characters is governed by the mode into which the USART is programmed. For example, bits 6 and 7 of Mode Register 1 control the number of SYN characters used and the transparency mode. These registers are not used in this application.

6. Transmit/Receive Data Holding Registers

These two registers share the same address. When transmitting, the system CPU writes a data word to the Transmit Data Holding Register so that it can be serialized and transmitted. When receiving, the system CPU reads from the Receive Data Holding Register to receive the data word that has been received and deserialized. The detail of USART can be found in [Ref. 10: pp. 3.7 - 3.16].

CR-7	CR-6	CR-5	CR-4	CR-3	CR-2	CR-1	CR-0
operating mode		request to send	reset error	async: force break	receive control (RxEN)	data terminal ready	transmit control
00= normal operation 01= async:auto echo mode sync:syn and/or DLE stripping mode 10=local loop back 11= remote loop back		0=forces RTS output high 1=forces RTS output low	0=normal 1=reset error flag in status register	0=normal 1=force break sync: send DLE 0=normal 1=send DLE	0= disable 1= enable	0=force DTR output high 1=force DTR output low	0= disable 1= enable

Figure 2.5 Command Register Format.

SR-7	SR-6	SR-5	SR-4	SR-3	SR-2	SR-1	SR-0
data set ready	data carrier detect	FE/SYN detect	overrun	PE/DLE detect	TxEMT/ DSCHG	RxRDY	TxRDY
0=DSR input high 1=DSR input low	0=DCD input high 1=DCD input low	async: 0=normal 1= framing error sync: 0=normal 1=SYN char detected	0= normal 1= overrun error	async: 0= normal 1=parity error sync: 0=normal 1=parity error or DLE char received	0=normal 1=change in DSR or DCD, or Tx shift register is empty	0=Rx holding reg is empty 1=Rx holding reg has data	0=Tx holding reg busy 1=Tx holding reg is empty

Figure 2.6   Status Register Format.

III. SYSTEM SOFTWARE

An operating system is any program or group of related programs whose purpose is to act as an intermediary between the hardware and the user of a computer. In the case of a small microcomputer application, the operating system may consist of little more than a set of service routines and a simple set of commands that allows the user to load or dump memory to or from an external storage device, to modify bytes, and to test application programs.

A more sophisticated operating system may have a disk-controller interface and some form of memory management. The highest level of operating system includes such features as multiprogramming, bank-selected memory, a hard-disk controller interface, and random track-sector allocation. CP/M (Control Program for Microcomputers) is typical of a class of operating systems for the microcomputers currently in widespread use.

This chapter describes the structure of CP/M-85 operating system for the Z-100 microcomputers and some modifications to it.

A. CP/M-85 OPERATING SYSTEM

1. General Discussion

CP/M-85 operating system for the Z-100 microcomputer is divided into two software components: the "system kernel" and the "BIOS files".

The system kernel is a set of programs that reside on the reserved system tracks of a disk. Special data transfer utilities (usually SYSGEN and/or MVCPM207) are used to copy the system kernel from one disk to another. The system kernel manages files, translates the commands you enter at the keyboard, and performs other functions that do not depend upon specific hardware characteristics.

CP/M-85 BIOS consists of two files, BIOS85.SYS and BIOS88.SYS, which reside on the file tracks of the disk. The portion of the BIOS stored in the file BIOS85.SYS uses the 8085 processor of the Z-100. The portion stored in BIOS88.SYS uses the 8088 processor of the Z-100. For further information on the CP/M-85, refer to [Ref. 7: pp 1.9 - 1.10].

2. Structure

CP/M is logically divided into four parts, called the Basic I/O System (BIOS), the Basic Disk Operating System (BDOS), the Console Command Processor (CCP), and the Transient Program Area (TPA). The memory organization of the CP/M system is in the Figure 3.1.

a. Basic Disk Operating System (BDOS)

The BDOS is responsible for all management functions including disk-file management, I/O high-level management, and all other function calls available to the user. The BDOS makes calls to the BIOS, which interfaces with the actual hardware environment of the microcomputer. For the detail of the BDOS functions,see [Ref. 12: p. 140]. and [Ref. 8: pp. 124 - 125].

Our SPOOL program will use some of the BDOS functions:List Output, Make File, Write File, Close File, and so on. The List Output function is activated by pressing Control-P key, which copies everything that appears on the screen to the printer.

Access to a file on the disk is done through an File Control Block (FCB). An FCB is supplied for each file the user wishes to access. The detailed description of FCB is found in [Ref. 8: pp. 93 - 94].

b. Basic I/O System (BIOS)

The BIOS is a hardware-dependent module that defines the exact low level interface with a particular computer system that is necessary for peripheral device I/O. The BIOS is a collection of user-written subroutines for primitive character I/O and disk access. Disk functions include: set DMA address, set track and sector, read sector, write sector and so on. A series of jump vectors is located at the head of the BIOS and points to all internal subroutines. The BDOS locates subroutines in the BIOS by calling the subroutines at the known location in the jump vector area.

c. Console Command Processor (CCP)

When CP/M becomes active, it gives control to the CCP, which, is actually an application program loaded at the top of the TPA. When first entered, the CCP displays a prompt character, 'A>'. This indicates which disk drive is currently logged in for use. To change drives, the user simply enters the drive name and a colon (for example, B:). This causes the CCP to request the BDOS function call "select disk," with register E containing a number representing the requested drive (A=1, B=2, and so forth). There are five built-in commands: TYPE, DIR, REN, ERA, and SAVE. A number of transient commands are also provided: SYSGEN, PIP, ED, ASM, DUMP, LOAD, STAT, MOVCPM, SUBMIT, and DDT. These commands are actually .COM files or memory image files. For the detail of the CCP operation, see [Ref. 12: pp. 148 - 149].

d. Transient Program Area (TPA)

The TPA is an area of memory (i.e., the portion that is not used by the FDOS and CCP) where various nonresident operating system commands and user programs are executed. The lower portion of memory is reserved for system information. For the detail about the memory organization, see [Ref. 8: pp 89 - 90].

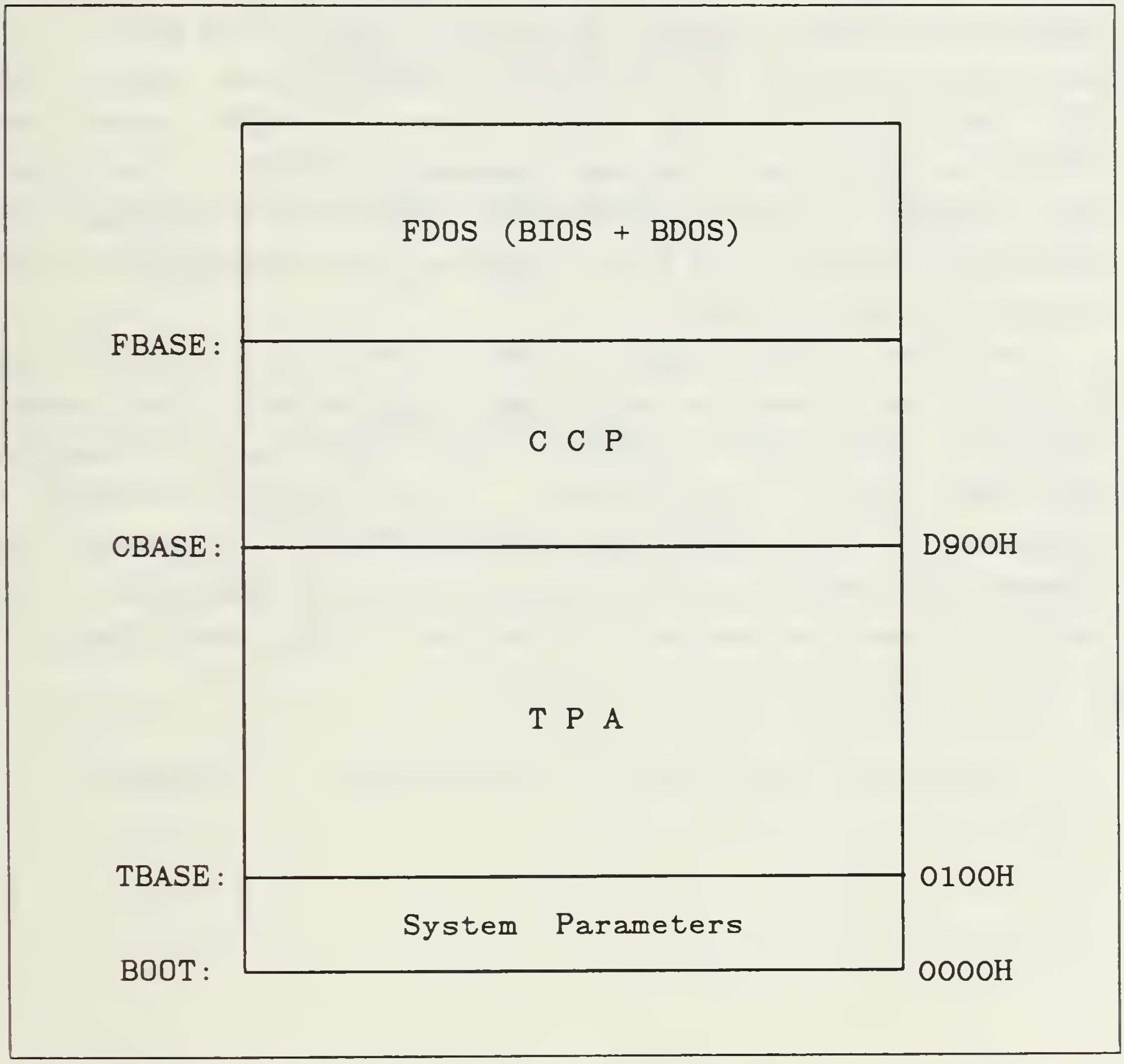


Figure 3.1 Memory Organization of CP/M.

B. MODIFICATION TO CP/M-85 BIOS

Since the SPOOL utility program uses the List Output function as described in the previous section, we should intercept some point of the List Output function routine in BIOS. The BIOS85.SYS file could be modified by using the DDT debugging utility.

The BIOS entry point of List Output function is located at F93DH. This is shown in the Figure 3.2 a. We patch this point so that the List Output function pass through a certain intercept program in the TPA area to the BIOS. It's necessary to place the intercept routine at an area of the TPA that is near to the BIOS. Since the BIOS starts from D900H, we place the interrupt program at D800H. The interrupt program is contained in the SPOOL utility program. When the user activates the SPOOL program, the intercept routine is also loaded at D800H.

The List Output function is activated by pressing the Control-P key. When the user press the Control-P key, everything on the screen is catched and loaded into E register and then sent to the printer. The intercept routine is intended to intercept the character in E register and provide the SPOOL utility program with that character. The detail of this mechanism is discussed in the next chapter.

```
F93D   MVI  A,05

F93F   JMP  F99C

F942   MVI  A,06
```

(a) Original Entry Point of BIOS

```
F93D   JMP  D800

F940   NOP

F941   NOP

F942   MVI  A,06
```

(b) Modified Entry Point of BIOS

Figure 3.2 Entry Point of List Output Function.

IV. IMPLEMENTATION OF THE SYSTEM

This chapter describes the structure of the Concentrator, the connection between the Concentrator and Z-100's, and the logic designs of three processes. In the last section of the chapter, we address the integrated operation of the processes.

A. STRUCTURE OF THE CONCENTRATOR

The structure of the Concentrator is shown in Figure 4.1. The Concentrator consists of one iSBC 86/12A single board computer and three BLC 8538 eight channel I/O expansion boards. They are connected to the MULTIBUS so that the single board computer could access any channel of the three expansion boards. Since each expansion board has eight channels, the Concentrator can have twenty four channels. Each channel of the expansion board could be connected to one peripheral device. Therefore, the Concentrator can control twenty four peripheral devices. In our design, the printer is connected to the Channel zero, and the other twenty three channels are connected to the TTY ports of Z-100's. Each channel is associated with a Universal Synchronous Asynchronous Receiver Transmitter(USART), which contains four registers: Data, Status, Mode 1, Mode 2, and Command registers. The details about programming the USART are in the Chapter 2. For more information about the address assignment of the registers, see [Ref. 10: pp. 3.3 - 3.11].

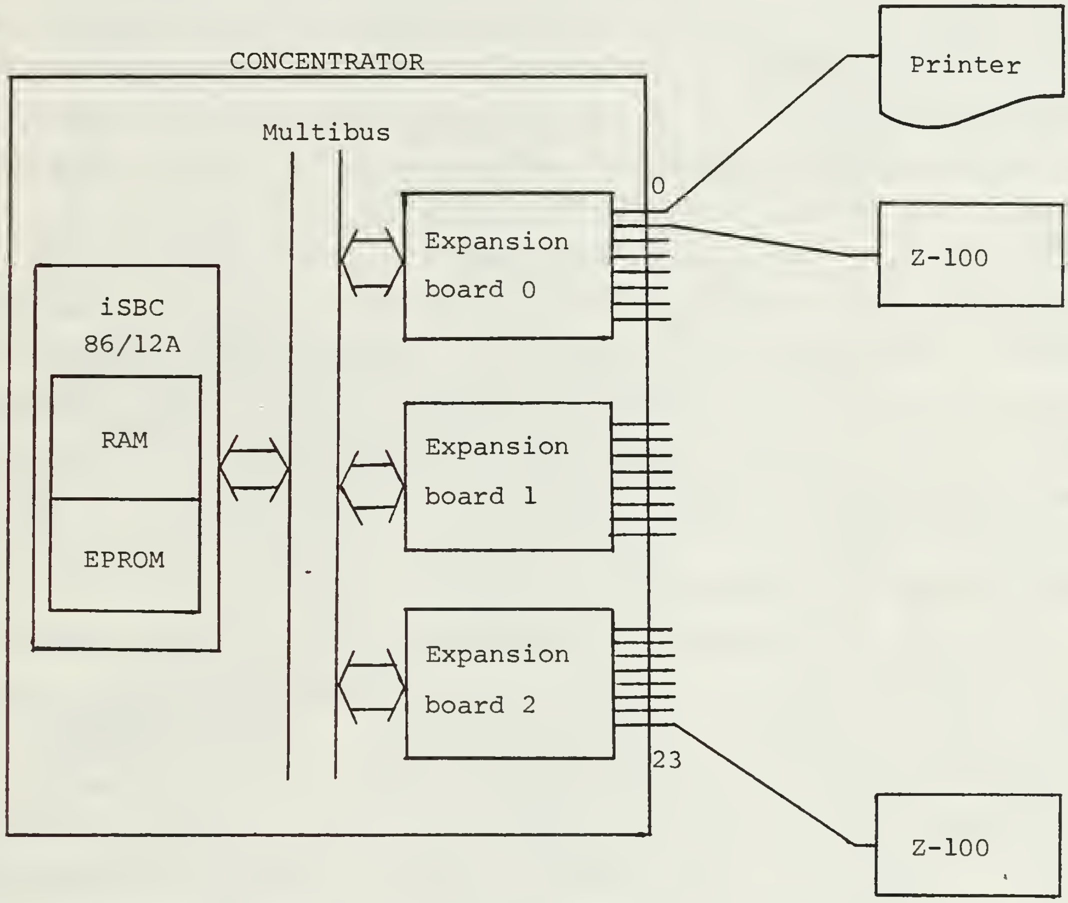


Figure 4.1 Structure of the Concentrator.

B. BOOT PROCESS

The function of the BOOT process is to download the CONTROL process from any Z-100 into the RAM of the single board computer. The logical flow of the BOOT process is shown in Figure 4.2 and the algorithm of the BOOT process is in Figure 4.3. When the Concentrator is turned on, the BOOT process inside the EPROM of the iSBC 86/12A starts to check the status ports of the multi-channel expansion boards in a round robin fashion to check if a Z-100 is turned on or not. Since the CONSOLE of Z-100 is assigned to the TTY port, the Z-100 will send a message terminating with 'A>' to the dedicated port of the expansion board when it is turned on. The BOOT process will send a command 'TYPE A:CONTROL.CMD' to down-load the CONTROL process when it reads in the 'A>'. Then, the Z-100 will send the CONTROL.CMD file to the Concentrator while the BOOT process reads in the file and stores it in the RAM. Because the TYPE function of the CP/M operating system stops printing the file when it meets a Control-Z character, we have taken another character, the percentage '%', to signal the end of file.

On the completion of downloading the CONTROL.CMD file, the CONTROL process is executed. The CONTROL process issues the reset command 'STAT CON:=CRT:', which changes the CONSOLE device from TTY to CRT. Then, 'A>' will appear on the screen of the Z-100 and the user of the Z-100 can do his work. From now, the CONTROL process starts to communicate with the printer and the multiple Z-100 microcomputers. The program listing of the BOOT process is in APPENDIX A.

C. CONTROL PROCESS

The function of the CONTROL process is to control the printer and multiple Z-100's and to connect the printer to any one of the multiple Z-100's at a time so that the data could be transferred from Z-100 to the printer. The purposes of the CONTROL process are:

1) To allow use of the printer for any connected Z-100.

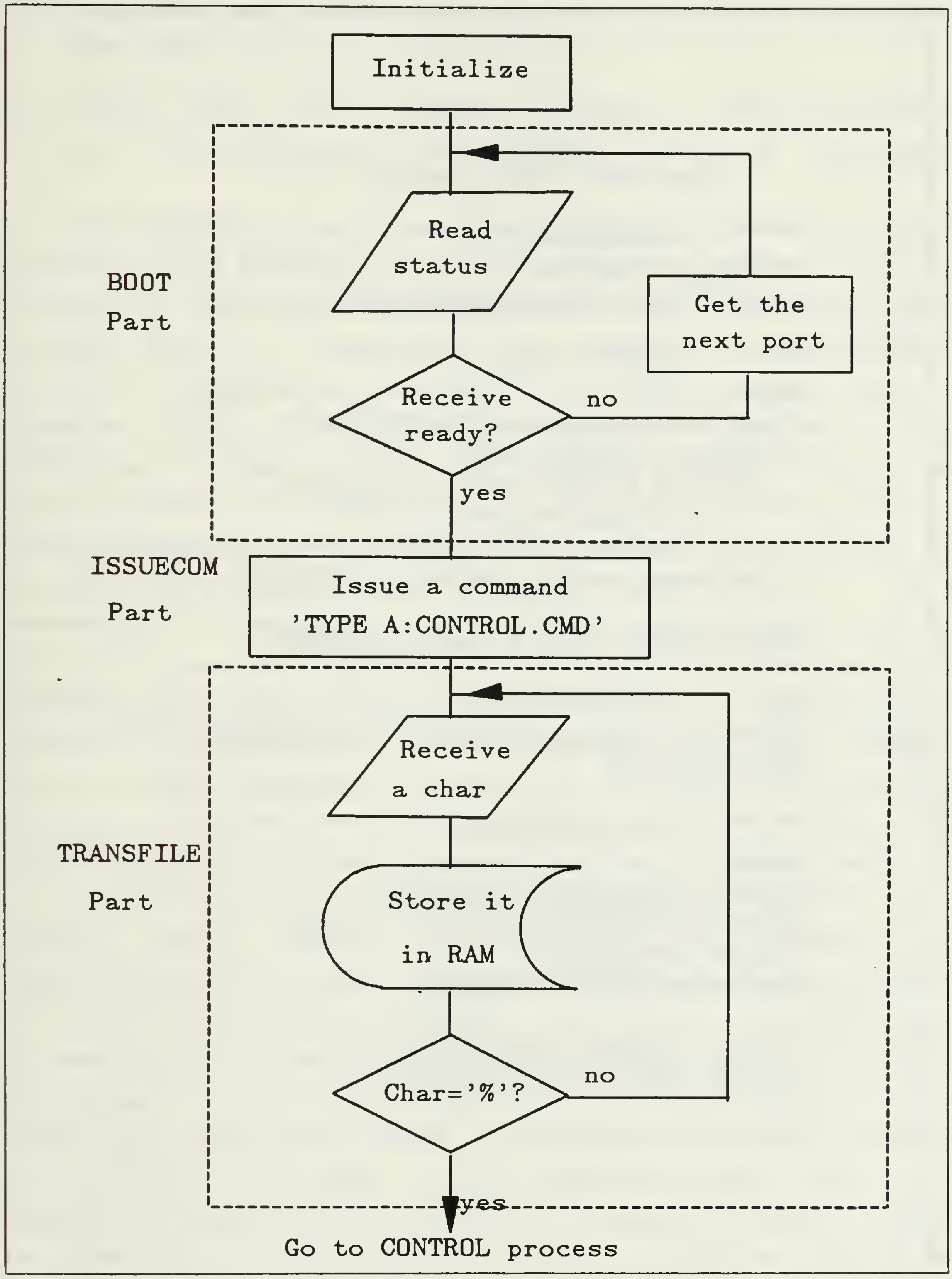


Figure 4.2 Logical Flow of BOOT.

```
        Algorithm of BOOT Program

*****       Initialization       *****
Initialize the USARTs
Set PORTNO with the first status port

*****       BOOT     Part       *****
Repeat
  Read the status port given by PORTNO
  If the port is Receive-ready
     then read a character from the data port
          If the character is '>'
             then go to ISSUECOM part
     else store the next status port into PORTNO

Until the control is sent to the ISSUECOM

*****       ISSUECOM  Part       *****
Set the command string with 'TYPE A:CONTROL.CMD'
Call COMMAND

*****       TRANSFILE Part       *****
Repeat
  Read a character from the data port
  Store the character into the new code area
Until the character is '%'
Execute the CONTROL program

*****    Subroutine  COMMAND    *****
Send the command string to the Z-100
```

Figure 4.3 Algorithm of BOOT.

2) To permit any Z-100 to be activated, even during the time that the printer is in use.

The logical flow of the CONTROL process is shown in Figure 4.4 and the algorithm of the CONTROL process is in Figure 4.5.

The CONTROL process is stored in CONTROL.CMD file of all Z-100's. The CONTROL.CMD file, however, is downloaded into the RAM of the single board computer when the Concentrator and the first of the Z-100's is turned on. The CONTROL process is composed of two states: the control state and the transfer state. The CONTROL process lies in one of two states at a time. When the CONTROL process is in the control state, it checks the status ports of the multi-channel expansion boards in a round robin fashion to see if any boot or print request is issued from a Z-100. If a channel has a request, the CONTROL process reads in the character from the data port of the channel. The character sequence 'Q' means a print request and 'A>' means a boot request. On receiving the character 'Q', the CONTROL process sends a confirm character 'Y' to the Z-100 and then goes to the transfer state. In the transfer state, the CONTROL process receives a character from Z-100 and sends it to the printer using a software handshaking method. The CONTROL process checks all the other ports whenever one hundred and twenty eight characters are sent to the printer to know if they want to boot-up. For every port which wants to boot-up, it issues the command 'STAT CON:=CRT:', and then continues to print the characters in the transfer state. After that, the Z-100 which has received the reset command can work its own job. The CONTROL process stops its transfer of data between the Z-100 and printer when it reads in a Control-Z (end-of-file) character, and then enters the control state to check the next status port.

When the CONTROL process reads in 'A>' in the control state, it issues a reset command 'STAT CON:=CRT:' because

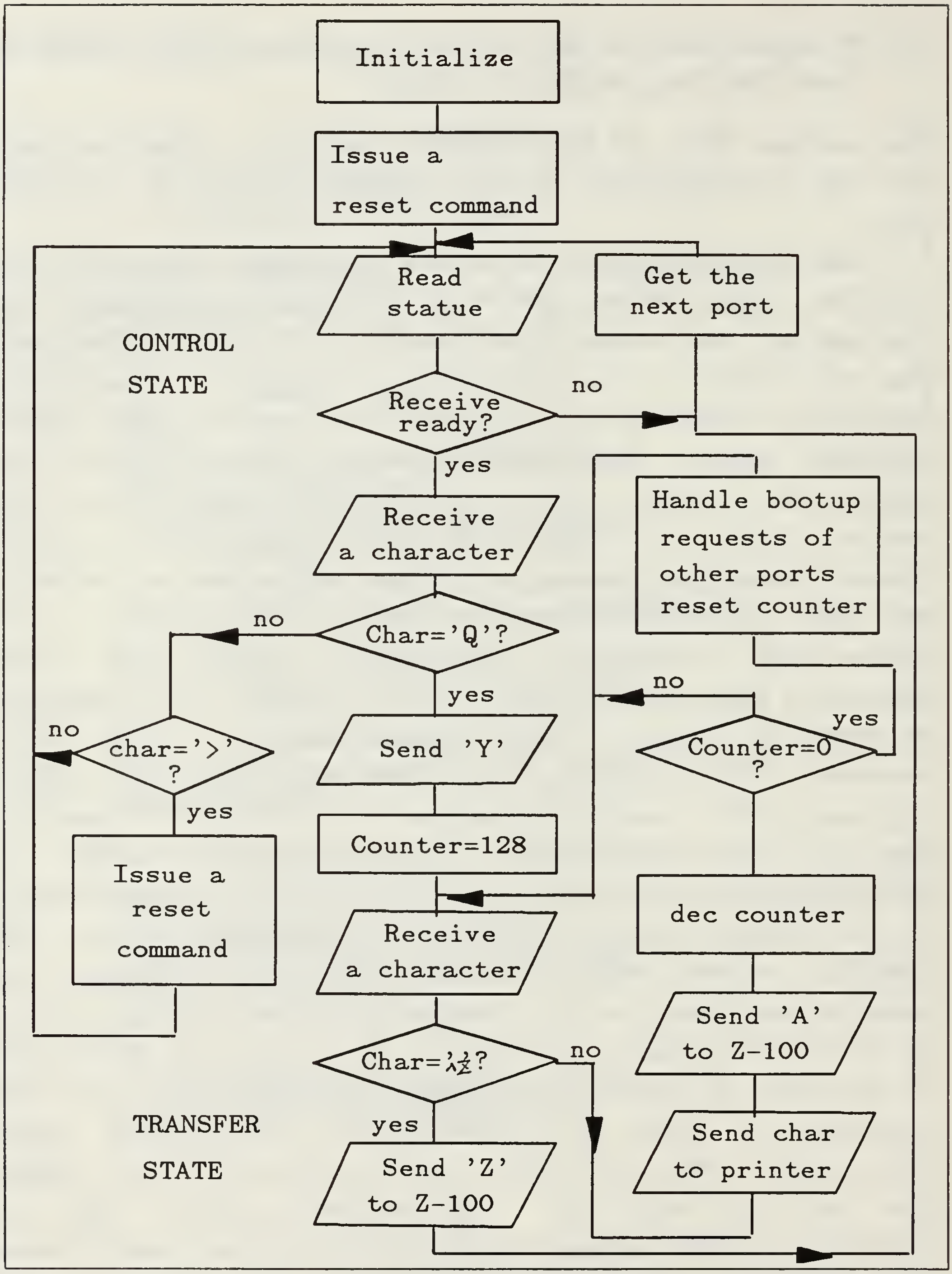


Figure 4.4 Logical Flow of CONTROL.

```
Algorithm of CONTROL Program

*****        Initialization        *****
Set PORTNO with the first status port
Call RESET

*****        CONTROL   Part        *****
Repeat
  Read the status port given by PORTNO
  If the port is receive-ready
    then read a character from  the data port
       If the charater is '>'
         then call RESET
         else if the character is 'Q'
              then go to TRANSFER
    else store the next status port into PORTNO
Until the system control is sent to the TRANSFER

*****        TRANSFER   Part        *****
Repeat
  Read a character from the data port of Z-100
  Send the character to the printer port
  Call BCHECK on every 128 characters
Until meet the End-Of-File mark '^Z'
Go to CONTROL

*****     Subroutine COMMAND      *****
Send the command string to the Z-100

*****     Subroutine BCHECK      *****
Check all other status ports once
For each port which has the boot-up request
call RESET
```

Figure 4.5   Algorithm of CONTROL.

there is no need for downloading the CONTROL.CMD file. On receiving other characters than 'Q' or 'A>', the CONTROL process ignores them and goes to the next port to check.

Program listing of CONTROL process is in APPENDIX B.

D. SPOOL PROCESS

The SPOOL process has three functions. It also has the intercept routine for intercepting the List Output function of the BIOS. The intercept routine is really a part of the List Output function of the BIOS. The intercept routine receives the control from the List Output function and calls the SPOOL program, and then returns the control back to the List Output function of the BIOS. The mechanism between the List Output function and the SPOOL program is explained below.

The SPOOL process has three options, they are:

1. Send the characters on the screen of a Z-100 directly to the printer.
2. Save the characters on the screen of a Z-100 into a disk file.
3. Print any existing disk file.

The logical flow of the SPOOL process is shown in Figure 4.6 and the algorithm of the SPOOL process is in Figure 4.7.

The user can select any one of the above options by typing in the appropriate digit corresponding to the selection. Each function is implemented with a pair of modules: Set and Path. Function 1 is composed of Set1 and Path1, Function 2 is composed of Set2 and Path2, and Function 3 is of Set3 and Path3.

The entry point on the head of the SPOOL program is composed of a selection routine which jumps to one of 3 Path modules depending on the user's selection number.

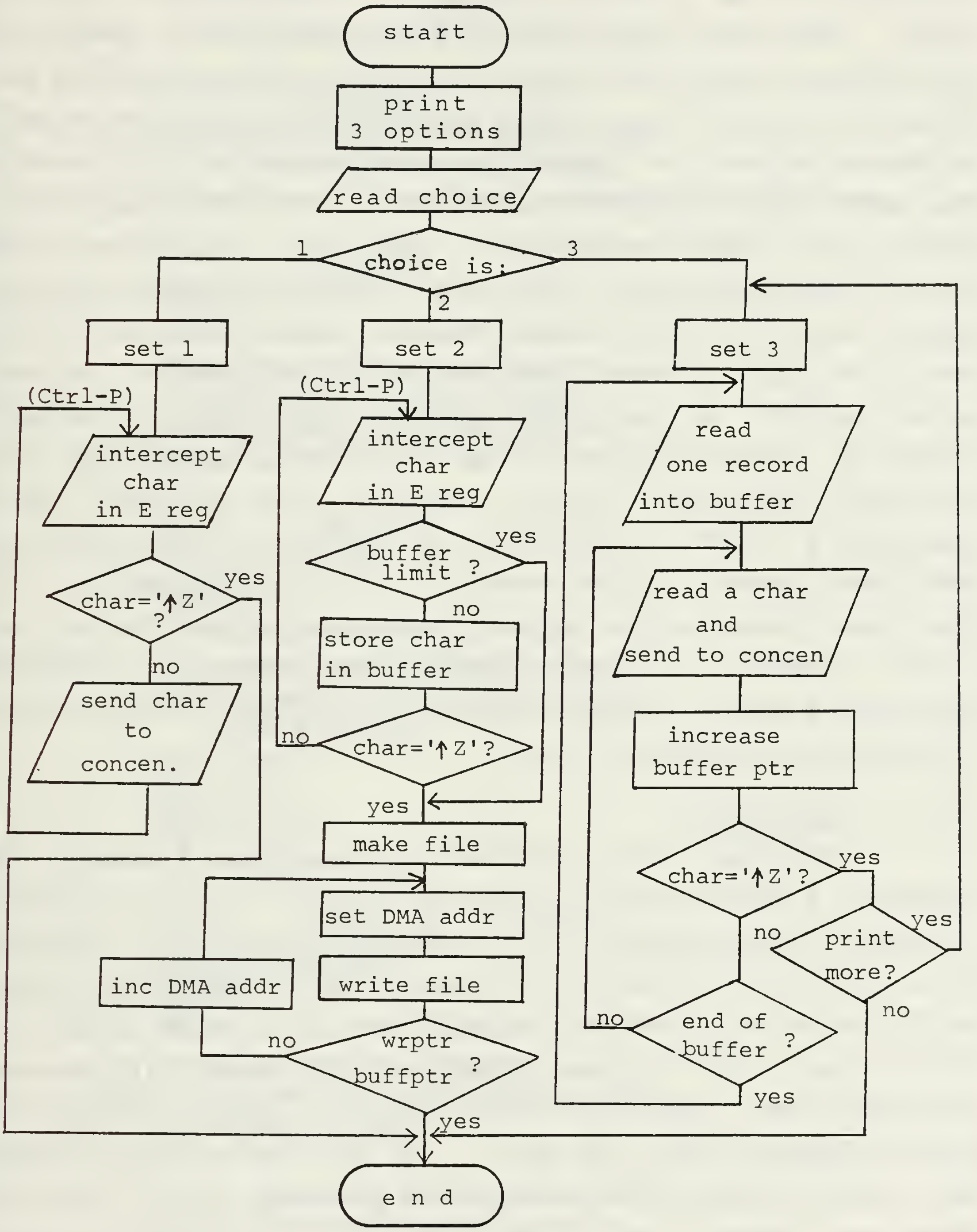


Figure 4.6 Logical Flow of SPOOL.

1. Function 1

Function 1 sends the character on the screen directly to the printer through the Concentrator in Ctrl-P mode. When the user selects Function 1, Set1 module sets the buffer pointer with its own buffer address and the path-flag with number one. Then the buffer pointer contains the starting address of the buffer whose size is 128 bytes and the path-flag contains number one. On the completion of setting the buffer pointer and path-flag, Function 1 comes out to the CCP level, printing a message that user should press the Ctrl-P key and go ahead. As the user hits Ctrl-P key, the List Output routine calls the address of entry point (D000H) of the SPOOL process whenever a character is typed in. Since the path-flag is set with number one, Path1 is then activated. Path1 reads in the character in the register E which is captured by the List Output function, and then sends the character to the Concentrator. The Function 1 returns the control back to the List Output function. Function 1 repeats this process until the user hits the Ctrl-Z key. When the user hits the Ctrl-Z key, Function 1 comes out to the CCP level.

2. Function 2

Function 2 is similar to Function 1 except that Function 2 saves the character in a disk file while Function 1 sends the character to the printer directly. When the user selects Function 2, Set2 module sets the buffer pointer with the buffer address and the path-flag with number one. Set2, also, reads in user-specified file name. If, however, user does not specify his/her own file name, default file name (TEMPFILE.$$$) is used. On the completion of setting the buffer pointer, path-flag, and reading in the file name, Function 2 comes out to the CCP level, printing a message that the user should press the Ctrl-P key and go ahead. As the user hits Ctrl-P key, the List Output routine calls the address of entry point, which activates Path2. Path2 reads

```
        Algorithm of SPOOL

*****     Initialization      *****
The user selects one of the three options
Go to the corresponding SET module
Each SET module goes to the coresponding PATH module

*****        PATH 1           *****
Repeat
  Read a character from the Z-100
  Store the character into the buffer
  Send the character to the printer
Until it meets the End-Of-File mark '^Z'
Go to the CCP level

*****        PATH 2           *****
Call READNAME
Repeat
  Read a character from the Z-100
  Store the character into the buffer
Until it meets the End-Of-File mark '^Z'
      or the buffer limit
Make a file
Write the file
Close the file
Go to the CCP level

*****        PATH 3           *****
Call READNAME
Repeat
  Read a record from the file
  Send the record to the printer
Until it meets the End-Of-File mark '^Z'
Go to the CCP level

*****  Subroutine READNAME  *****
Read a file name from the console
```

Figure 4.7   Algorithm of SPOOL.

in the character in register E which is captured by the List Output function, and then store it in the buffer while the buffer pointer is incremented. And then Function 2 returns the control back to the List Output function. Path2 repeats this process until the user hits the Ctrl-Z key. Path2 stores the Ctrl-Z character on the end of the buffer when it meets the Ctrl-Z character. After that, Path2 makes a file and write the file with the content of buffer by one record(128 bytes) at a time. We use 2 pointers, the write pointer and the buffer pointer, during writing the file. Buffer pointer points to the address of the last character in the buffer while write pointer is set with the starting address of the buffer. Whenever a record is written in the file, the write pointer is increased by 80H. Write process is done when the write pointer becomes greater than the buffer pointer. Since the buffer size is not large enough for a big file, when the buffer pointer exceeds the limit Function 2 will issue a message that the buffer is full and the user may take an action to reset the pointers.

3. Function 3

Function 3 reads a file whose file name is given by user and sends the contents of file to the printer through the Concentrator. Function 3 reads in one record of the file at a time and repeats this until it meets the Ctrl-Z character. The buffer is the same one of Function 1. Once reading one record into the buffer, Function 3 sends the content of buffer one byte at a time to the printer thru Concentrator. Function 3 does not need the Ctrl-P function while function 1 and 2 use it. Program listing of SPOOL process is in APPENDIX C.

V. CONCLUSIONS

The primary goal of this thesis was met. The CONTROL process was designed for the communication with the multiple Z-100 microcomputers and the printer, and this is successfully integrated into the Printer Multiplexing System with the SPOOL utility program. The BOOT process was designed for downloading the CONTROL process from any one of the Z-100's, which eliminated the need for the disk drive, console and/or additional EPROM to store the CONTROL process.

This system allows use of the printer for any connected Z-100, permits interactive use of the printer to one user when nobody else is waiting to use it, and enables the effective interactive sharing of the printer among multiple users by spooling any character on the screen into a disk file for later batch printing.

Future research involving the Printer Multiplexing System should develop the 8086 code of the SPOOL utility program so that the users could use the SPOOL in the environment of CP/M-86 or MS-DOS. The BOOT and CONTROL processes are already written in the 8086 code.

Additionally, we are expecting that this system could be integrated into the Ethernet. Then the CONTROL process should be expanded to contain the Ethernet protocol program.

APPENDIX A

USER MANUAL FOR PRINTER SHARING

A. SYSTEM CONFIGURATION

The Printer Sharing system consists of one iSBC 86/12A single board computer, three BLC 8538 I/O expansion boards, the multiple Z-100 microcomputers, and the printer. The detail about the iSBC 86/12A single board computer and the BLC 8538 I/O expansion boards can be found in the Chapter 2.

B. ACTIVATING THE SYSTEM

Turn on the Concentrator before any Z-100 microcomputer is turned on. The user should keep this power-up procedure because the BOOT program needs to initialize the USART before checking the ports of the I/O expansion boards. At this time, the CONSOLE device of the Z-100 is normally assigned to the teletype port. Otherwise, the user should change the CONSOLE device of the Z-100 computer to the teletype port manually. And then, the CONTROL.CMD file in drive A is down-loaded into the RAM of the single board computer. When the CONTROL program begins to execute, it changes the CONSOLE device to the standard device CRT. Then, the Z-100 user can do his own work on the computer.

C. USING THE SPOOL PROGRAM

To use the printer, the user should type 'SPOOL' in the CCP level. Then there appears a guidance about the three options of the SPOOL program on the screen. The user can select any one of the three options by typing in the corresponding number of that option. The three options are listed below.

1. Interactive Print

When the user selects option 1, he can use the printer interactively. After the user press the Ctrl-P key, everything on the screen is copied to the printer through the Concentrator. This option ends when the user press the Ctrl-Z key.

2. Saving on a Disk File

When the user selects option 2, he can save the characters on the screen on the disk file. The user is asked to type in the file name. If the file name is not typed in, the default file name (TEMPFILE.$$$) is used. After the user press the Ctrl-P key, everything on the screen is saved in the buffer. If the buffer size exceeds the limit, a warning message appears on the screen. In this case, the user should take an action as directed on the screen. This option ends when the user press the Ctrl-Z key, writing the buffer into the disk file.

3. Printing Existing Files

When the user selects option 3, he can print any existing file. The user is asked to type in the file name the user wants to print. After a file is printed out, another file can be printed continuously if the user wants to do.

APPENDIX B

BOOT PROGRAM

```
;***********************************************************
;*                                                         *
;*                     BOOT  PROGRAM                       *
;*                                                         *
;*             Dec 1985.   By Choi and Lee                 *
;***********************************************************
;BOOT program reads in the status ports of the I/O expansion
;boards in a round robin fashion. If there is a boot request
;on the port, the BOOT  program  down-loads the  CONTROL.CMD
;file  from  the  Z-100  microcomputer.  And then, the BOOT
;prgram sends the control to the CONTROL program.


            cseg
            org    0


;***********************************************************
;*                  INITIALIZATION                         *
;***********************************************************
;Initialize  the  USART  and  a  variable  which  contains
;the port number.  On the completion of initialization, the
;control goes to the BOOT process.

            mov   al,0ffh
            out   0a0h,al
            mov   dh,0

            mov   dl,82h
            mov   al,4eh
            call  init

            mov   dl,82h
            mov   al,3eh
            call  init

            mov   dl,83h
            mov   al,07h
            call  init

            mov   portno,85h
            mov   dh,0


;*****************************************************************
;*                        BOOT  PART                             *
;*****************************************************************
;Receives  '>' character from  any one of the multiple Z-100
;microcomputers and  then downloads the CONTROL.CMD  file to
;the memory of iSBC 86/12A. On the completion of loading the
;file, it transfers the control  to the  CONTROL  program.


boot:      mov   dl,portno   ;load DL reg with port number
           in    al,dx       ;read the status of the port
           and   al,02       ;receive ready?
           jz    nextport    ;if not go to next port
           dec   dx          ;get the data port
           in    al,dx       ;read in the character from Z-100
           inc   dx          ;get the status port
           cmp   al,'>'      ;compare the character with '>'
           jnz   nextport    ;this Z-100 is not booted up,
                             ;go to nextport
           jmp   issuecom    ;go to ISSUECOM process

nextport:  add   dl,4        ;get the next status port
           cmp   dl,0A0h     ;check if out of range
```

```
          jbe    bypass1      ;if inside the range, come along
                              ;the normal path
          mov    dl,85h       ;if out of range,set DL reg with
                              ;the 1st status port

bypass1:  mov    portno,dl    ;store  addr of  next status port
                              ;in the variable.
          jmp    boot         ;repeat checking the status ports



;************************************************************
;*                 ISSUE COMMAND  PART                      *
;************************************************************
;Issue the command "TYPE A:CONTROL.CMD" to the Z-100


issuecom: mov    si,0                  ;set counter with 0
          mov    bx,offset storage     ;set base reg with addr
                                       ;of command string
          call   command               ;send the TYPE command



;************************************************************
;*             TRANSfer  control.cmd  FILE                  *
;************************************************************
;Download the control.cmd file from Z-100 to the memory   of
;iSBC 86/12A. On the completion of  downloading the  CONTROL
;file, the TRANSFILE sends the control to CONTROL program.


transfile:mov    bx,offset newcode   ;set base reg with   addr
                                     ;of newcode area
          mov    si,0                ;set counter with 0

nextchar: in     al,dx        ;read in status port
          and    al,02        ;receive ready?
          jz     nextchar    ;if not,read again
          dec    dx          ;get data port
          in     al,dx       ;read a char from Z-100
          cmp    al,19h      ;compare with temporary Ctrl-Z
          jnz    bypass2     ;if not,come along normal path
          inc    al          ;recover Ctrl-Z to 1AH

bypass2:  mov    (bx+si),al;store char in new code area
          cmp    al,'%'      ;compare with end of file
          jz     adjust      ;go to adjust
          inc    si          ;increase counter
          inc    dx          ;get status port
          jmp    nextchar


;************************************************************
;*                        ADJUST                            *
;************************************************************
;Reads in  the  prompt character  'A>' from Z-100, and  then
;goes to  the CONTROL program.

adjust:   inc    dx          ;get the status port
          mov    cl,4        ;set the counter

statin1:  in     al,dx       ;check if receive ready
          and    al,02
          jz     statin1
          dec    dx          ;get the data port
          in     al,dx       ;read a char from Z-100
          inc    dx          ;get the status port
          dec    cl          ;decrease the counter
          jnz    statin1     ;if not done, read again
          jmp    newcode + 132h  ;execute the CONTROL process
```

```
;***********************************************************
;*          subroutine     Initialization                 *
;***********************************************************
;Initialize the USART.


init:      out   dx,al   ;write programming code to USART
           add   dl,4    ;get the next port
           cmp   dl,0A0h ;check if out of range
           jbe   init    ;if not,write to the next port
           ret


;***********************************************************
;*              subroutine     COMMAND                    *
;***********************************************************
;Sends the command string to the Z-100


command:   in    al,dx       ;check if transmit ready
           and   al,01
           jz    command
           mov   al,(bx+si) ;move a char from command string
           dec   dx          ;get data port of Z-100
           out   dx,al       ;send char to Z-100
           inc   dx          ;get status port
           cmp   al,0dh      ;compare char with end_of_string
           jz    eocom       ;if matches,end of command

statin:    in    al,dx       ;check if receive ready
           and   al,02
           jz    statin
           dec   dx          ;get data port of Z-100
           in    al,dx       ;read a response from Z-100
           inc   dx          ;get status port
           inc   si          ;increase counter
```

```
          jmp    command     ;send another char


eocom:    in     al,dx       ;check if receive ready
          and    al,02
          jz     eocom
          dec    dx          ;get data port of Z-100
          in     al,dx       ;read a response from Z-100
          inc    dx          ;get status port
          ret


;************************************************************
;*                 NEW    CODE    AREA                      *
;************************************************************


newcode:
;************************************************************
;*                 DATA   STORAGE   AREA                    *
;************************************************************


          dseg
          org    300h

storage   db     'type a:control.cmd'
          db     0dh

portno    db     0
          end
```

APPENDIX C

CONTROL PROCESS

```
;************************************************************
;*                                                          *
;*                    CONTROL   PROGRAM                     *
;*                                                          *
;*             Dec 1985.   By  Choi  and  Lee               *
;************************************************************
;CONTROL program communicates with the printer and  multiple
;Z-100 microcomputers. The CONTROL program resides in either
;the control state and transfer state.


          cseg

prtdata   equ   80h

prtstat   equ   81h

          org   0


;************************************************************
;*                      INITIALIZATION                      *
;************************************************************
          mov   portno+20eh,85h ;set the port # to 85h
          mov   dh,0             ;set DH to 0
          call  reset            ;let the Z-100 CONSOLE gain
                                 ;control
```

```
;**************************************************************
;*                       CONTROL   PART                       *
;**************************************************************
control:  mov   dl,portno+20eh ;load DL reg with 1st port
          in    al,dx         ;read the status port
          and   al,02         ;check if any request is out
          jz    nextport      ;if no, go to the next port
          dec   dx
          in    al,dx         ;read the data port
          inc   dx
          cmp   al,'Q'        ;verify the request
          jz    transfer      ;enter transfer state
          cmp   al,'>'
          jnz   nextport
          call  reset         ;issue a reset command

nextport: add   dl,4          ;get the next port
          cmp   dl,0a0h       ;check if out of range
          jbe   bypass1       ;if not,come along normal path
          mov   dl,85h

bypass1:  mov   portno+20eh,dl ;store the increased port #
          jmp   control


;**************************************************************
;*                     TRANSFER  PART                         *
;**************************************************************


transfer: in    al,dx
          and   al,01
          jz    transfer
          mov   al,'Y'     ;send "yes" character to Z-100
          dec   dx
          out   dx,al
```

```
         mov   cl,80h      ;set the char counter with 128

return:  inc   dx

statin1: in    al,prtstat
         and   al,01
         jz    statin1

statin2: in    al,dx
         and   al,02
         jz    statin2
         dec   dx
         in    al,dx       ;read in the data of Z-100
         inc   dx          ;get the status port
         cmp   al,19h      ;compare with Ctrl-Z(1AH). 1AH  is
                           ;temporarily changed  because TYPE
                           ;function stops printing at Ctrl-Z
         jz    eotrans     ;if so, go to end of transfer
         out   prtdata,al;send the data to the printer

statin3: in    al,dx
         and   al,01
         jz    statin3
         dec   dx
         mov   al,'A'      ;send an 'Acknowledge'
         out   dx,al
         dec   cl          ;decrease the char counter
         jnz   return
         call  bcheck      ;check all other ports to know  if
                           ;there is any boot-up request
         jmp   return
```

```
;*************************************************************
;*                     END OF TRANSFER                       *
;*************************************************************
;End of Transfer  sends the  ending signal to the  Z-100 and
;a carriage return and a line feed character to the printer.
eotrans:  in    al,dx      ;check if Z-100 is transmit ready
          and   al,01
          jz    eotrans
          mov   al,'Z'     ;load AL reg with ending signal
          dec   dx         ;get data port
          out   dx,al      ;send ending signal to Z-100

statin4:  in    al,prtstat;check if printer is ready
          and   al,01
          jz    statin4
          mov   al,0dh     ;send a carriage return to printer
          out   prtdata,al

statin5:  in    al,prtstat;check if printer is ready
          and   al,01
          jz    statin5
          mov   al,0ah     ;send a line feed to printer
          out   prtdata,al
          jmp   control


;*************************************************************
;*              subroutine      RESET                        *
;*************************************************************
;RESET subroutine issues the command 'STAT CON:=CRT:' to the
;Z-100.

reset:    mov   si,0
          mov   bx,offset storage+20eh
          call  command
          ret
```

```
;*************************************************************
;*                 subroutine      COMMAND                   *
;*************************************************************
;Sends the command string to the Z-100

command:  in    al,dx        ;check if transmit ready
          and   al,01
          jz    command
          mov   al,(bx+si) ;move a char from command string
          dec   dx           ;get data port of Z-100
          out   dx,al        ;send char to Z-100
          inc   dx           ;get status port
          cmp   al,0dh       ;compart char with end_of_string
          jz    eocom        ;if matches,end of command

statin6:  in    al,dx        ;check if receive ready
          and   al,02
          jz    statin6
          dec   dx           ;get data port of Z-100
          in    al,dx        ;read a response from Z-100
          inc   dx           ;get status port
          inc   si           ;increase counter
          jmp   command      ;send another char


eocom:    in    al,dx        ;check if receive ready
          and   al,02
          jz    eocom
          dec   dx           ;get data port of Z-100
          in    al,dx        ;read a response from Z-100
          inc   dx           ;get status port
          ret
```

```
;*************************************************************
;*                    subroutine BOOTCHECK                  *
;*************************************************************
;Checks  all the  other  status ports  to know if there is a
;boot-up request from the Z-100.  For  each  port  which has
;a boot-up request,issues a reset command.

bcheck:    inc    dx             ;get the current status port
           mov    ch,dl          ;store it in CH register
           mov    dl,85h         ;starts from the first port

ncheck:    cmp    dl,ch          ;check if the port is itself
           jnz    bypass3
           add    dl,4

bypass3:   cmp    dl,0a0h        ;check if out of range
           ja     eocheck        ;if yes,end of checking

           in     al,dx          ;read the status port
           and    al,02          ;check if any request is out
           jz     nextport1      ;if no, go to the next port
           dec    dx
           in     al,dx          ;read the data port
           inc    dx
           cmp    al,'>'
           jnz    nextport1
           call   reset          ;let Z-100 console gain control

nextport1:add     dl,4           ;get the next port
           jmp    ncheck         ;check if out of range

eocheck:   mov    dl,ch          ;recover the current status
           dec    dx;port
           mov    cl,80h         ;reset the char counter
           ret
```

```
;*************************************************************
;*                  DATA  STORAGE  AREA                      *
;*************************************************************
          dseg
          org 100

portno    db    0

storage   db    'stat con:=crt:'
          db    0dh
          db    '%'
          db    lah
          end
```

APPENDIX D

SPOOL PROCESS

```
;***********************************************************
;*                                                         *
;*                     SPOOL  PROGRAM                      *
;*                                                         *
;*              Dec 1985.  By  Choi  and  Lee              *
;***********************************************************
;SPOOL program has three options:print interactively,   save
;the char into a disk file, and print any existing file.  It
;also contains  the intercept  routine  for intercepting the
;char in the E register in the Ctrl-P mode.


;***********************************************************
;*                    DECLARATION  PART                    *
;***********************************************************


conin     equ   1       ;console input char

printf    equ   9       ;print string

readconf  equ   10      ;read console buffer

openf     equ   15      ;open file

closef    equ   16      ;close file

readf     equ   20      ;read file

writef    equ   21      ;write file

makef     equ   22      ;make file

setdmaf   equ   26      ;set DMA address
```

```
eof         equ   1ah      ;end_of_file character

ttystat     equ   0e9h     ;TTY  status port

ttydata     equ   0e8h     ;TTY  data   port


boot        equ   0000h    ;warm boot entry

bdos        equ   0005h    ;bdos entry

fcb         equ   005ch    ;default file control block

dbuff       equ   0080h    ;default DMA address

wrptr       equ   607ch    ;write buffer pointer

pointer     equ   607eh    ;temporary DMA buffer pointer

tdma        equ   6080h    ;temporary DMA buffer address



;************************************************************
;*                     INITIALIZATION                       *
;************************************************************
            org   100h
            call  init


            org   0d000h


entry:            ;entry point of SPOOL process
            lda   pathfl     ;read the user's choice
            cpi   1          ;if option 1
            jz    path1      ;then,go to path1
            cpi   2          ;if option 2
            jz    path2      ;then,go to path2
            jmp   comerror   ;jmp to communication error
```

```
;************************************************************
;*                    module   PATH 1                       *
;************************************************************
;Catch  a  character  from  E register  and  store it in the
;buffer, and then send the character to the concentrator

path1:    push a!  push b!  push d!  push h!
          lhld  pointer   ;load HL reg with next buffer ptr
          mov   m,e       ;move input char to the buffer
          call  sendchar  ;send char to concentrator
          inx   h         ;increase buffer ptr
          mov   a,l       ;check if buffer ptr reached   end
          ani   0ffh      ;of buffer
          jnz   leap      ;if not,go along the normal path
          lxi   h,tdma    ;if end of buffer,reset buffer ptr

leap:     shld  pointer   ;store the increased buffer ptr
          pop h!  pop d!  pop b!  pop a!
          ret




;************************************************************
;*                    module  PATH 2                        *
;************************************************************
;Catch  a  character from  E register  and store it into the
;buffer. When the end_of_file(Ctrl-Z) character is met, make
;a file and write the buffer into the file

path2:    push a!  push b!  push d!  push h!
          lhld  pointer   ;load HL reg  with next buffer ptr
          mov   m,e       ;move input char to the buffer
          mov   a,e
          cpi   ' '       ;compare with end_of_file mark
          jz    save      ;if it matches,write the buffer
          cpi   '-'
          jnz   mark
          lda   warn
          cpi   1
          jz    warning

mark:     inx   h          ;increase buffer pointer
          mov   a,h
          cpi   0c0h
          jnz   markl
          mvi   a,l
          sta   warnfl

markl:    shld  pointer    ;store increased buffer pointer
          pop h!  pop d!  pop b!  pop a!
          ret




;**************************************************************
;*                   module  PATH 3                          *
;**************************************************************
;Read the file whose file name is given by the user into the
;buffer by one record(128 bytes) at a time.   And then send
;the characters of the buffer to the concentrator.

path3:    call  open      ;open the file
          lxi   d,nofile
          inr   a         ;if it fails to open the file
          cz    finis     ;go to finish with error message


copy:     call  read      ;read a record from file into buff
          ora   a         ;if it meets end_of_file char
          jnz   eotranf   ;go to end of transfer file
          call  filesend  ;send record to printer thru concen
          jmp   copy

eotranf:  lxi   d,normal ;load the address of normal message

finis:    call  trnsrdy  ;check if transmit ready
          mvi   a,eof    ;load Acc with end_of_file char
          out   ttydata  ;send it to concen
          call  rcvrdy   ;check if receive ready
          in    ttydata  ;read a response from concen
          mvi   c,printf ;print the normal message
          call  bdos


          call  morefile ;handle a request of another print
          jmp   boot     ;come out to CCP level




;************************************************************
;*                subroutine  SENDCHARACTER                 *
;************************************************************
;Send a character of the buffer to the concentrator

sendchar:
          call  trnsrdy  ;check if transmit ready
          mov   a,e      ;move char from E to Acc
          cpi   ' '      ;compare with ' ' of Ctrl-Z
          jz    eosend   ;if yes,then go to end of send
          out   ttydata  ;if not Ctrl-Z,send the char
          call  rcvrdy   ;check if receive ready
          in    ttydata  ;receive an acknowledgement char
          ret
```

```
;************************************************************
;*                         SAVE                             *
;************************************************************
save:     call  diskcopy
          call  eocopy




;************************************************************
;*                  subroutine  DISKCOPY                    *
;************************************************************
;Write the content of the buffer onto the disk file

diskcopy: lhld  pointer
          mvi   a,eof
          mov   m,a
          call  make       ;make a file

dmachg:   call  setdma     ;set DMA address
          call  write      ;write the DMA buffer onto the file


;Increase the write pointer by 80h and restore it
          lhld  wrptr      ;load HL reg with write ptr
          mov   a,l        ;move the lower addr of write ptr
          aci   80h        ;add 80h to it with carry
          mov   l,a        ;load L with increased lower addr
          jnc   leapl      ;if no carry,jmp to leapl
          mov   a,h        ;if carry is out,
          adi   1          ;add 1 to upper addr of write ptr
          mov   h,a        ;load H with increased upper addr

leapl:    shld  wrptr      ;restore the increased write ptr
```

```
;Compare the write pointer with the buffer pointer
          lhld  pointer   ;load HL reg with buffer ptr
          xchg            ;move HL reg to DE reg
          lhld  wrptr     ;load HL reg with write ptr
          mov   a,d       ;move upper addr of buffer ptr
          sbb   h         ;subtract write ptr from buffer ptr
          cm    eocopy    ;if result is minus,go to eocopy
          jp    dmachg    ;if result is plus,write more
          mov   a,e       ;move lower addr of buffer ptr
          sbb   l         ;subtract write ptr from buffer ptr
          jp    dmachg    ;if result is plus,write more
          ret



;**************************************************************
;*                 subroutine  FILESEND                       *
;**************************************************************
;This subroutine sends the characters of the DMA buffer   to
;the concentrator. This subroutine  returns the control back
;to the PATH3  when  it meets the  Ctrl-Z (end of file char)
;or the end of buffer.

filesend:
          mvi   c,128     ;set the buffer counter
          lxi   d,dbuff   ;set DE register with buffer addr

nextchar: ldax  d         ;load Acc with a char from buffer
          inx   d         ;increase buffer ptr
          mov   b,a       ;move the char of A into B

          call  trnsrdy   ;check if transmit ready
          mov   a,b       ;get stored char
          cpi   eof       ;compare the char with end_of_file
          jz    eotranf   ;if yes,then go to end of transfer
```

```
        out    ttydata   ;send the char to concen.

        call   rcvrdy    ;check if receive ready
        in     ttydata   ;receive an ack char from Z-100
        dcr    c         ;decrease buffer ptr
        jnz    nextchar  ;if not zero,get the next char
        ret



;***************************************************************
;*               subroutine  END OF SENDCHAR                   *
;***************************************************************


eosend: call   trnsrdy   ;check if transmit ready
        mvi    a,eof     ;yes,send an end_of_file char
        out    ttydata
        call   rcvrdy    ;check if receive ready
        in     ttydata   ;receive an ack char from concen.
        jmp    boot      ;come out to CCP


;**************  subroutine  END OF COPY  *****************

eocopy: call   close
        mvi    c,printf
        lxi    d,eosave
        call   bdos
        jmp    boot
```

```
;**************   subroutine  WARNING   *********************
warning:  mvi   a,0
          sta   0ec0dh
          mvi   c,printf
          lxi   d,warnmsg
          call  bdos
          call  diskcopy
          call  eocopy
          ret


;**************   subroutine  MAKE    ***********************

make:     mvi   c,makef
          lxi   d,tfcb
          call  bdos
          ret


;**************   subroutine  OPEN    ***********************

open:     mvi   c,openf
          lxi   d,tfcb
          call  bdos
          ret

;**************   subroutine  READ    ***********************

read:     mvi   c,readf
          lxi   d,tfcb
          call  bdos
          ret
```

```
;**************    subroutine  CLOSE    **********************

close:     mvi   c,closef
           lxi   d,tfcb
           call  bdos
           ret


;**************    subroutine  WRITE    **********************

write:     mvi   c,writef
           lxi   d,tfcb
           call  bdos
           ret


;**************    subroutine  SETDMA    *********************

setdma:    mvi   c,setdmaf
           lhld  wrptr
           xchg
           call  bdos
           ret


;**************    subroutine  TRANSMIT READY    *************

trnsrdy:   in    ttystat
           ani   01
           jz    trnsrdy
           ret


;**************    subroutine  RECEIVE  READY    *************

rcvrdy:    in    ttystat
           ani   02
           jz    rcvrdy
           ret
```

```
;****************************************************************
;*                subroutine   INITIALIZATION                   *
;****************************************************************
;Select an option of three alternatives

init:
          mvi   c,printf  ;print the query string
          lxi   d,query
          call  bdos

          mvi   c,conin   ;read the answer of user
          call  bdos
          cpi   '1'
          cz    set1      ;print out directly in Ctrl-P mode
          cpi   '2'
          cz    set2      ;save on a file in Ctrl-P mode
          cpi   '3'
          cz    set3      ;print any existing file
          call  typerror  ;typing error
          ret




;****************************************************************
;*                       module  SET 1                          *
;****************************************************************
;Set the buffer pointer and path-flag.  Also  checks if  the
;printer is busy or not.

set1:     lxi   h,tdma    ;load HL reg with addr of buffer
          shld  pointer   ;store buffer addr into buffer ptr
          mvi   a,1       ;set pathflag with number 1
          sta   pathfl

          call  trnsrdy   ;check if transmit ready
          mvi   a,'Q'     ;send a request to concen.
          out   ttydata

          mvi   c,0ffh    ;set counter with # of trials

statinl:  in    ttystat   ;read status port
          dcr   c         ;decrease the counter
          cz    waite     ;if zero,should wait
          ani   02
          jz    statinl
          in    ttydata   ;receive an ack from concen.

          mvi   c,printf  ;print a starting message
          lxi   d,start
          call  bdos
          jmp   boot      ;come out to CCP




;***********************************************************
;*                   module   SET 2                        *
;***********************************************************
;Set the buffer pointer and write pointer.    Also construct
;the FCB whose file name is given by the user.

set2:     lxi   h,tdma    ;set the buffer and write ptr with
          shld  pointer   ;the buffer address
          shld  wrptr
          mvi   a,2       ;set the pathflag with number 2
          sta   pathfl

          mvi   c,printf  ;print the selection message
          lxi   d,selname ;load DE reg with addr of msg
          call  bdos
          call  readname  ;read file name given by the user

          mvi   c,printf  ;print the starting message
```

```
            lxi    d,start
            call   bdos
            jmp    boot




;*************************************************************
;*                  module  SET 3                            *
;*************************************************************
;Construct the FCB whose file name is given by the user, and
;check if the printer is busy or not.

set3:       mvi    c,printf  ;print file name message
            lxi    d,selname ;load DE reg with addr of message
            call   bdos

            call   readname  ;read the filename typed by   user
                             ;into FCB
            call   trnsrdy   ;check if transmit ready
            mvi    a,'Q'     ;send a request to concen.
            out    ttydata

            mvi    c,0ffh    ;set counter with # of trials

statin2:    in     ttystat   ;read status port
            dcr    c         ;decrease the counter
            jz     waite     ;if zero,printer is busy
            ani    02        ;check if receive ready
            jz     statin2   ;if not ready,read again
            in     ttydata   ;read response from concen.
            cpi    'Y'       ;is response 'Yes'?
            jnz    comerror  ;if not,communication error
            jmp    path3     ;starts to send the file
```

```
;*****************************************************************
;*              Subroutine    MORE    FILE                       *
;*****************************************************************
morefile:
         mvi    c,33      ;set counter with # of char of AFCB
         lxi    d,afcb    ;load DE reg with addr of AFCB
         lxi    h,tfcb    ;load HL reg with addr of TFCB

movafcb: ldax   d         ;load Acc with char of AFCB
         mov    m,a       ;store char of Acc into TFCB
         inx    d         ;increasd addr of AFCB
         inx    h         ;increase addr of TFCB
         dcr    c         ;decrease counter
         jnz    movafcb   ;if not done, move the nest char

         mvi    c,printf ;print the message
         lxi    d,moreprt
         call   bdos
         mvi    c,conin   ;read the answer of user
         call   bdos
         cpi    'Y'       ;if yes,print anotherfile
         cz     set3
         cpi    'N'       ;if no,come out to CCP
         cnz    typerror ;if any other char, error message
         ret


;****************     Subroutine  TYPERROR    ****************

typerror:
         mvi    c,printf ;print error message
         lxi    d,typerrm
         call   bdos
         jmp    boot
```

```
;*********     Subroutine   COMmunication  ERROR   *********

comerror:
         mvi    a,'P'     ;tell   the   concentrator  of
                          ;communication error
         out    ttydata
         mvi    c,printf  ;print error message on the screen
         lxi    d,comerrm
         call   bdos
         jmp    boot


;**************     Subroutine    WAITE    ******************

waite:
         mvi    a,'P'     ;tell the concen. of communication
                          ;error
         out    ttydata
         mvi    c,printf  ;print the wait message
         lxi    d,wait
         call   bdos
         jmp    boot



;**************     Subroutine    READNAME    ***************

readname:
         mvi    c,readconf    ;read the edited file name
         lxi    d,filename    ;given by user
         call   bdos

         lxi    d,filename+1  ;load DE addr of char counter
         ldax   d             ;load Acc the # of chars
         cpi    0             ;if file name is not typed,
         jz     default       ;then take the default filename
```

```
          mov   c,a            ;move the # of chars to C reg
          inx   d              ;load DE reg with the address of
                               ;name buffer
          lxi   h,tfcb+1     ;load HL reg with the addr of TFCB

          mvi   b,8         ;set the # of chars of the filename

readfnr:
          ldax  d              ;move the filename to TFCB
          cpi   '.'            ;if it meets the delimiter,
          jz    readft         ;then read filetype
          call  charconv       ;convert the lower case into
                               ;upper case character
          mov   m,a
          inx   d
          inx   h
          dcr   c
          jz    eoread         ;no filetype, go to end_of_read
          dcr   b              ;if # of filename exceeds 8
          jz    meetdel        ;then, go to meet_delimiter
          jmp   readfnr        ;read the next char of filename


meetdel:
          ldax  d              ;read a char from console buffer
          cpi   '.'            ;compare with the delimitter
          jz    readft         ;if matches,read file type
          inx   d              ;increase name buffer pointer
          jmp   meetdel        ;repeat reading char


readft:   mvi   b,3            ;set char counter with 3
          inx   d              ;increase name buffer pointer
          dcr   c              ;decrease # of chars
          lxi   h,tfcb+9       ;load HL with addr of file type
                               ;of File Control Block
```

```
readftr: ldax  d              ;read a char from name buffer
         call  charconv       ;convert lower case into upper
         mov   m,a            ;store char into FCB
         dcr   c              ;decrease # of chars
         jz    eoread         ;if name buffer exhausted
                              ;then,go to end of read
         dcr   b              ;decrease file type counter
         jz    eoread         ;if FCB is full,go to eoread
         inx   d              ;increase name buffer pointer
         inx   h              ;increase FCB pointer
         jmp   readftr

eoread:
         ret


;***************   Subroutine   DEFAULT   *****************
;copy the default FCB into Temporary FCB.

default:
         mvi   c,33       ;set the counter with 33
         lxi   d,dfcb     ;set Default FCB pointer
         lxi   h,tfcb     ;set Temporary FCB pointer


movfcb:  ldax  d          ;read a char from DFCB
         mov   m,a        ;store it into TFCB
         inx   d          ;increase DFCB pointer
         inx   h          ;increase TFCB pointer
         dcr   c          ;decrease counter
         jnz   movfcb     ;move another char
         ret
```

```
;*************    Subroutine   CHARCONVert    **************
;Converts the characters from 'a' through 'z' into the upper
;case characters.

charconv:
        cpi    'a'
        jm     nochange
        cpi    'z'
        jp     nochange
        sui    20h

nochange:
        ret


;************************************************************
;*                 DATA      AREA                          *
;************************************************************

pathfl: db      0

warn:   db      0

dfcb:   db      0
        db      'TEMPFILE'
        db      '$$$'
        db      0,0,0,0
        ds      16
        db      0
        ds      3


tfcb:   db      0
        db      '        '
        db      '   '
        db      0,0,0,0
        ds      16
        db      0
        ds      3

afcb:   db      0
        db      '        '
        db      '   '
        db      0,0,0,0
        ds      16
        db      0
        ds      3


filename:
        db      16
        ds      16


query:  db      0dh,0ah
        db      'Welcome to SPOOL utility. ',0dh,0ah,0dh,0ah
        db      'Three options are: ',0dh,0ah

        db     '1. Print any characters on console directly'
        db     'to the printer.'
        db      0dh,0ah
        db    '2. Save any charecters on console in a file.'
        db     0dh,0ah
        db     'Print any existing file.',0dh,0ah
        db     0dh,0ah,0dh,0ah
        db     'Select any one of three options above : $'


start:  db     0dh,0ah
        db     'Press the Control_p key and go ahead.'
        db     0dh,0ah
        db     'On the completion of your work, press the'
        db     ' Control_Z key'
        db     0dh,0ah,'$'

selname:db     0dh,0ah,'File name : $'
```

```
moreprt:db      0dh,0ah,'Do you like to print another file?'
        db      '(Y/N) : $'

typerrm:db      0dh,0ah
        db      'Typing error. Try again.$',0dh,0ah,'$'

eoprint:db      0dh,0ah,'Printing complete.',0dh,0ah,'$'

eosave: db      0dh,0ah,'Saving file complete.',0dh,0ah

wait:   db      0dh,0ah,'Printer is busy now. Try again.'
        db      0dh,0ah,'$'

comerrm:db      0dh,0ah,'Protocol error. Try again.'
        db      0dh,0ah,'$'

nofile: db      0dh,0ah,'No source file.',0dh,0ah,'$'

wrprot: db      0dh,0ah,'Write protected?',0dh,0ah,'$'

normal: db      0dh,0ah,'Spool complete.',0dh,0ah,'$'

warnmsg:db      0dh,0ah,'Your file is out of buffer size'
        db      0dh,0ah,'Make your file again',0dh,0ah,'$'
        ds      32

stack:


;*****************************************************************
;*              Intercept Routine of Control-p                   *
;*****************************************************************

        org     0d800h
        call    0d000h
        mvi     a,19h
        jmp     0f99ch

        end
```

LIST OF REFERENCES

1. Perry, Mark L., Logic Design of a Shared Disk System in a Multi-Micro Computer Environment, M.S. Thesis, Naval Postgraduate School, June 1983.

2. Dwyer, Thomas A. and Chritchfield, Margot CP/M and the Personal Computer, Addison Wesley Inc., April 1984.

3. Morgan, Christopher L. and Waite, Mitchell, 8086/8088 16-Bit Microprocessor Primer, Mc Graw Hill Inc., 1982.

4. INTEL Corporation, ISBC 86/12A Single Board Computer Hardware Reference Manual, 1979.

5. Digital Research, CP/M-86 Operating System Programmer's Guide, January 1983.

6. Digital Research, CP/M-86 Operating System System Guide, 1981.

7. Digital Research, CP/M-85 Software Documentation, 1982.

8. Digital Research, CP/M Operating System Manual, 1982.

9. INTEL Corporation, INTELLEC Microcomputer Development System Hardware Reference Manual, 1976.

10. National Semiconductor Corporation, BLC 8534/8538 4 and 8 Channel Communication Expansion Boards Hardware Reference Manual, June 1981.

11. Comfort, W. J. III, Console I/O Routines, September 1984.

12. Dahmke, Mark, Microcomputer Operating Systems, McGraw-Hill Book Company,1982.

13. Money, S.A., Microprocessor Data Book, McGraw-Hill Book Company,1981.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5100	2
3.	Department Chairman, Code 52 Department of Computer Sciences Naval Postgraduate School Monterey, California 93943-5100	1
4.	Computer Technology Department, Code 37 Department of Computer Sciences Naval Postgraduate School Monterey, California 93943-5100	1
5.	Prof. Uno R. Kodres, Code 52Kr Department of Computer Sciences Naval Postgraduate School Monterey, California 93943-5100	2
6.	Inst. Linda C. Rawlinson, Code 52Rv Department of Computer Sciences Naval Postgraduate School Monterey, California 93943-5100	1
7.	Directorate of EDPS Department of Comptroller Staff Army Headquarters, 140-01, Seoul, Republic of Korea	1
8.	Central Computer Center Army Headquarters, 140-01, Seoul, Republic of Korea	1
9.	Major Lee, Jang Hun 681 Palm Avenue Apt #A Seaside, California 93955	1
10.	Major Jang, Chang Ki SMC 2623 NPGS Monterey California 93943	1
11.	Captain Choi,Kwang Jun 851-62 Mia-7 Dong, Dobong Ku, Seoul, Republic of Korea	7
12.	Captain Lee,Ju Kab Military Apt 2-209 Etaewondong Yongsan Ku, Seoul, Republic of Korea	8
13.	Captain Kang, Seon Mo SMC 1988 NPGS Monterey California 93943	1
14.	LT. Ham, Byung Woon Naval Academy, Jinhae City, Gyungnam, Seoul, Republic of Korea	1

602- 197

216177

Thesis
C448854
c.1

Choi
Printer multiplexing among multiple Z-100 microcomputers.

16 MAY 90
80353

216177

Thesis
C448854
c.1

Choi
Printer multiplexing among multiple Z-100 microcomputers.

thesC448854
Printer multiplexing among multiple Z-10
3 2768 000 64758 0
DUDLEY KNOX LIBRARY